NFA035 – Exemple de sujet d'examen

juin 2013

Durée : 3h – Documents autorisés – Barème indicatif

Exercice 1 *Entrées/sorties 7 points*

Indication : pour cet exercice, nous ne pensons pas qu'il soit utile d'utiliser `StreamTokenizer`.
On veut écrire des méthodes pour afficher la table des matières d'un fichier texte au format décrit ci-après.

On suppose que le fichier lu est au format suivant :
- Les titres de niveau 3 (sous-sous-titres) sont sur une ligne séparée dont les trois premiers caractères sont `'***'` ;
- Les titres de niveau 2 (sous-titres) sont sur une ligne séparée dont les deux premiers caractères sont `'**'` (attention, une ligne qui commence par "***" commence aussi par deux "*", mais c'est un titre de niveau 3 et pas de niveau 2).
- Les titres de niveau 1 sont sur une ligne séparée dont le premier caractère est `'*'` (évidemment les titres de niveau 3 et 2 sont exclus) ;
- Les autres lignes sont du texte normal.

Par exemple le fichier ci-dessous à gauche a pour table des matière le texte ci-dessous à droite.

Contenu du fichier

```
* Programmation
  La programmation c'est bien.
  Ça ouvre l'esprit.
 ** Langages de programmation
  Les langages...
  *** Langage C
  Le langage C c'est bien car ça
  forme la jeunesse.
*** Langage Java
  Le langage Java...
** Compilation
 La compilation c'est mal connu.
* Système
Le système c'est bien aussi.
** Unix
Unix c'est bien
** Windows
Windows c'est moins bien
** Mac
Mac c'est pas mal
```

Table des matières

```
* Programmation
** Langages de programmation
*** Langage C
*** Langage Java
** Compilation
* Système
** Unix
** Windows
** Mac
```

Question 1.1 *2 points*

Écrivez la méthode `void tableDesMatieres(File f)` qui affiche les lignes qui commencent par "*" (y compris leur préfixes `'*'`, `'**'` et `'***'`) du fichier `f`.

Remarque : Il n'est pas permis d'utiliser la méthode `titres` ci-dessous.

Une correction possible :

On peut se poser la question de savoir si on doit attraper les exceptions ou non. En prenant le sujet au pied de la lettre, il faudrait les attraper.

```
public void tableDesMatieres(File f) {
 try {
  FileReader r0= new FileReader(f);
  BufferedReader r= new BufferedReader(r0);
  String l= r.readLine();
  while (l != null) {
    if (l.startsWith("*")) {
      System.out.println(l); // (ou Terminal.ecrireStringln(l);
    }
    l= r.readLine(); // ne pas oublier...
  }
  r.close();
 } catch (IOException e) {
   System.out.println("erreur...");
 }
}
```

Question 1.2 *4 points*

Écrivez la méthode `void titres(File f,int n)` qui affiche les lignes du fichier `f` qui sont des titres de niveau `n`.

Correction :

```
public void titres(File f, int n) throws IOException {
  String debut= "";
  for (int i= 0; i < n; i++) {
    debut+= "*";
  }
  FileReader r0= new FileReader(f);
  BufferedReader r= new BufferedReader(r0);
  String l= r.readLine();
  while (l != null) {
    // niveau n, et pas n+1...
    if (l.startsWith(debut) && ! l.startsWith(debut + "*")) {
      System.out.println(l); // (ou Terminal.ecrireStringln(l);
    }
    l= r.readLine(); // ne pas oublier...
  }
  r.close();
}
```

Remarque : si vous avez oublié que startsWith existait, on peut le reprogrammer avec une boucle assez simple...

Question 1.3 *1 points*

Question de cours : en utilisant la méthode `titres(File f,int n)` de l'exercice 1 (même si vous ne l'avez pas définie), écrivez la méthode `void titres(String s,int n)` qui affiche les lignes du fichier dont le nom est `s` qui sont des titres de niveau `n`.

Correction :

```
void titres(String s,int n) {
  File f = new File(s);
  titres(f,n);
}
```

Exercice 2 *collections (7 points)*

On souhaite modéliser le dossier de patients qui sont les clients d'une pharmacie. Chaque patient est représenté par son nom et sa liste de médicaments (chacun de type String) de son ordonnance, sans doublons. Les opérations sur le dossier des patients devront **garantir que les noms de patients sont différents, et qu'un médicament ne figure qu'une fois dans une ordonnance**. Une partie du code des classes Patient et `DossierPharmacie` vous est fournie : à vous de compléter le corps des méthodes signalées et de répondre aux questions posées.

Question 2.1 *classe Patient, 1 point*

Complétez le code pour la classe Patient donné plus bas.

```
private String nom;
private Set<String> ordonnance;

public Patient(String n){
  nom = n;
  ordonnance = new HashSet<String>();
}
public String getNom() { return nom;}

public boolean ordonnanceVide(){
      return ordonnance.isEmpty();
}
public void affiche(){
     Terminal.ecrireStringln(getNom());
     afficheOrdonnance();
}
/** Ajoute un medicament de nom m dans ordonnance   */
public void ajoutMedicament(String m) {
      ordonnance.add(m);
}

// A completer -> code methodes suivantes

/** Affiche l'ordonnance du patient       */
public void afficheOrdonnance(){
    // A completer
}
/** Teste si ordonnance contient un medicament m     */
public boolean contientMedicament(String m) {
```

```
    // A completer
  }
}
```

Correction

```
/**
 * @author aponte
 *
 */
public class Patient {

        private String nom;
        private Set<String> ordonnance;

    public Patient(String n){
        nom = n;
        ordonnance = new HashSet<String>();
    }
    public String toString() {
        return (nom);
    }

    public String getNom() { return nom;}

    /** Affiche donnes patient + ordonnance
     */
    public void affiche(){
        Terminal.ecrireStringln(toString());
        afficheOrdonnance();
    }

        // Question 1.2: A completer -> code  methodes suivantes
    /** Ajoute un medicament (sans doublons)
     * @param m: medicament                                --> 0.5
     */
    public void ajoutMedicament(String m) {
        ordonnance.add(m);
    }
    /** Affiche l'ordonnance du patient                --> 1
     */
    public void afficheOrdonnance(){
        for (String n: ordonnance){
                Terminal.ecrireStringln("  "+ n);
        }
    }
    /** Teste si ordonnance contient medicament      --> 0.5
     * @param m
     * @return
     */
    public boolean contientMedicament(String m) {
        return ordonnance.contains(m);
```

```
        }
        /** Teste si l'ordonnance de ce patient est vide
         *  necessaire pour la question 4
         */
        public boolean ordonnanceVide(){
            return ordonnance.isEmpty();
        }

        /** Methode employee pour calculer ensemble
         * de tous les medicaments demandes
         * Cette question a ete supprimee du sujet.
         */
        public Set<String> getOrdonnance(){
            return ordonnance;
        }
    }
```

Question 2.2 *classe DossierPharmacie, 3 points*

L'ensemble de patients clients de la pharmacie est représenté par une table d'associations entre noms de patients (String) et objets Patient. Les noms des patients doivent être tous différents, sans distinction entre minuscules et majuscules. Afin d'éviter l'ajout multiple d'un nom identique aux majuscules/minuscules près, l'ajout d'une nouveau nom dans la table se fait après avoir convertit celui-ci en minuscules (méthode `nouveauPatient`). Complétez le code des méthodes signalées plus bas.

```
public class DossierPharmacie {
  private String nom;
  private HashMap<String,Patient> patients;

  public DossierPharmacie(String n){
     nom=n; patients = new HashMap<String,Patient>();
  }

  /** Ajoute un nouveau patient de nom et ordonnance donnés */
  public void nouveauPatient(String nom, String [] ord){
     Patient c = new Patient(nom);
     for (String n: ord){
        c.ajoutMedicament(n);
     }
     String key=nom.toLowerCase();
     patients.put(key, c);
  }

  // Compléter les methodes suivantes

  /** Ajoute un nouveau medicament sur un patient deja existant.
   * Renvoie false si le patient n'existe pas et
   * true si l'ajout a pu etre effectue                    */
   public boolean ajoutMedicament(String nom, String m){
        // Completer
   }
   /** Affiche nom + ordonnance du patient du nom donné     */
```

```
    public void affichePatient(String nom){
       // Completer
    }
    /** Affiche nom  pharmacie + tous les patients du dossier   */
    public void affiche(){
         // Completer
    }
}
```

Correction

```
 /** Contraintes: pas de doublons de noms de patients,
  *  On ne distingue pas majuscule/minuscule pour comparer deux noms
  *  Les cles dans la mappe de contacts sont les noms en minuscules
  * @author aponte
  *
  */
 public class DossierPharmacie {

         private HashMap<String,Patient> patients;

         public DossierPharmacie(){
                 patients = new HashMap<String,Patient>();
         }

      /** Ajoute un nouveau patient de nom et medicaments donnes
       * @param nom nom patient
       * @param ord tableau de medicaments
       */
         public void nouveauPatient(String nom, String [] ord){
                 Patient c = new Patient(nom);
                 for (String n: ord){
                         c.ajoutMedicament(n);
                 }
                 String key=nom.toLowerCase();
                 patients.put(key, c);
         }

         // Compléter les methodes suivantes
         /** Ajoute un nouveu medicament sur
          * un patient deja existant.
          * Renvoie false si le patient n'existe pas et
          * true si l'ajout a pu etre effectue                (1 pt)
          */
          public boolean ajoutMedicament(String nom, String m){
                 Patient c =  patients.get(nom.toLowerCase());
                 if (c==null)
                         return false;
                 c.ajoutMedicament(m);
                 return true;
          }

         /** Affiche toutes les donnes et ordonnance du patient de nom
```

```
         * @param nom  --> 1
         */
        public void affichePatient(String nom){
                Patient c = patients.get(nom.toLowerCase());
                if (c==null)
                        Terminal.ecrireStringln("Aucun patient de nom "+nom);
                else
                        c.affiche();
        }
        /** Affiche  les patients du dossier  --> 1
         */
        public void affiche(){
                for (Patient c : patients.values()){
                        c.affiche();
                }
        }
        /** Retourne la collection de tous les patients qui prennent
         * @param m (medicament)    --> 1
         */
        public Collection<String> affichePatientAvecMedicament(String m){
                ArrayList<Patient> lc = new ArrayList<Patient>(patients.values());
                TreeSet<String> res = new TreeSet<String> ();
                for (int i = 0; i<lc.size(); i++){
                        Patient c = lc.get(i);
                        if (c.contientMedicament(m)) {
                                res.add(c.getNom());
                        }
                }
                return res;
        }


        // Question 5
        // Methode qui parcourt la mappe et enleve de la liste tous les patients d
        // l'ordonnance est vide.

        public void enleveOrdonnanceVide() {
                Set<Map.Entry<String, Patient>> s = patients.entrySet();
                Iterator<Map.Entry<String, Patient>> it = s.iterator();
                while (it.hasNext()) {
                        Patient p = it.next().getValue();
                        if (p.ordonnanceVide()){
                                it.remove();
                        }
                }
        }


}
```

Question 2.3 *(Patients ayant pris un médicament), 1,5 points*

Ajoutez dans la classe `DossierPharmacie` une méthode qui prend en paramètre le nom d'un médicament et retourne une collection contentant tous les patients de la pharmacie ayant pris ce médicament.

```
/** Retourne collection de patients qui prennent le medicament m    */
 public Collection<String> affichePatientAvecMedicament(String m){
       // Compléter
 }
```

Correction : voir ci-dessus.

Question 2.4 *Suppression de patients, 1,5 points*

Ajoutez dans la classe `DossierPharmacie` une méthode permettant de supprimer du dossier des patients tous les patients dont l'ordonnance est vide (aucun médicamant).

```
 /** Enleve du dossier tous les les patients dont
  * l'ordonnance est vide (ne contient aucun medicament)    */
 public void enlevePatientOrdonnanceVide() {
   // Compléter
```

Correction : voir ci-dessus.

Exercice 3 *5 points*

Un bout d'interface pour un logiciel de gestion des inscriptions.

On suppose donnée la classe CoursCnam. Un object CoursCnam représente un cours donné au Cnam. Il a un code (codeUE, comme "NFA035") et une durée en heures (40 pour NFA035 par exemple).

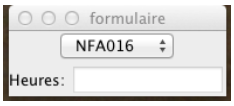
La liste de tous les cours est disponible à travers une méthode statique : `creerlesCours()`.

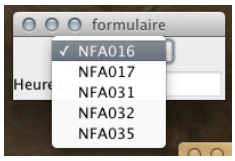
```
public class CoursCnam {
        private String codeUE;
        private int nombreHeures;

        /**
         * Méthode statique fabrique.
         * Retourne le tableau de tous les cours disponibles.
         * @return
         */
        public static CoursCnam[] creerLesCours() { ... }

        public CoursCnam(String codeUE, int nombreHeures) {...}

        public String getCodeUE() { return codeUE;}

        public int getNombreHeures() return nombreHeures;}

        public String toString() { return codeUE;}
}
```

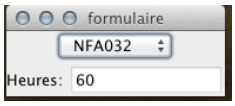
Question 3.1 *5 points*

Complétez la classe suivante en ajoutant le code nécessaire pour que, quand on sélectionne une UE dans la JComboBox, le nombre d'heures correspondantes s'affiche dans le champ texte `heuresField`. *Vous pouvez écrire des méthodes et des classes supplémentaires si vous le désirez.*

Exemple de l'interface en fonctionnement :







Indications :

- on peut être averti quand l'utilisateur sélectionne une entrée dans une `JComboBox` en utilisant un `ActionListener`.
- Pour fixer le modèle d'une JComboBox, on utilise la méthode `setModel(ComboboxModel model)`
- La classe `DefaultComboboxModel` a les constructeurs suivants :
  - `DefaultComboBoxModel()` Construit un modèle vide ;
  - `DefaultComboBoxModel(Object[] items)` construit un modèle à partir d'une liste de valeurs.
  - la méthode `getSelectedItem()` de `JCombobox` retourne *l'objet* sélectionné par l'utilisateur.
- le code demandé n'est pas très long.

```
public class Exam2 {
        private JComboBox ueListe = new JComboBox();
        private JTextField heuresChamp = new JTextField(10);
        private JFrame frame = new JFrame("formulaire");

        public Exam2() {
                placeComposants();
                activeComposants();
        }

        private void activeComposants() {
          // À ÉCRIRE
        }


        /**
         *  Place les composants (NE PAS ÉCRIRE!)
         */
        private void placeComposants() { ... }

        public static void main(String[] args) {
          // Lance le programme :
          // À ÉCRIRE
        }
}
```

Correction

```
public class Exam2 {
        private JComboBox ueListe = new JComboBox();
        private JTextField heuresChamp = new JTextField(10);
        private JFrame frame = new JFrame("formulaire");
```

```
        public Exam2() {
                placeComposants();
                activeComposants();
        }

        private void activeComposants() {
                ueListe.setModel(
                  new DefaultComboBoxModel(CoursCnam.creerLesCours())
                );
                ueListe.addActionListener(new ActionListener() {

                        public void actionPerformed(ActionEvent arg0) {
                                selectUE();
                        }
                });
        }

        protected void selectUE() {
                CoursCnam ue = (CoursCnam) ueListe.getSelectedItem();
                this.heuresChamp.setText(""+ue.getNombreHeures());
        }

        /**
         * Méthode pour placer les composants (ne pas écrire!)
         */
        private void placeComposants() {
                heuresChamp.setEditable(false);
                Container panel = frame.getContentPane();
                panel.setLayout(new GridBagLayout());
                GridBagConstraints c = new GridBagConstraints();
                c.gridwidth = 2;
                panel.add(ueListe, c);
                c = new GridBagConstraints();
                c.gridy = 1;
                panel.add(new JLabel("Heures: "), c);
                c.gridx = 1;
                panel.add(heuresChamp, c);
                frame.pack();
                frame.setVisible(true);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }

        public static void main(String[] args) {
                SwingUtilities.invokeLater(new Runnable() {

                        public void run() {
                                new Exam2();
                        }
                });
        }
}
```